Universidade Federal de Minas Gerais

Geração Adaptativa de Inimigos Usando Algoritmos Evolutivos em Jogos Digitais

Projeto Orientado em Computação II

Hiago de Souza Cruz Alves e Silva
Orientador: Prof. Dr. Luiz Chaimowicz

Resumo

Desenvolver e balancear inimigos é um processo custoso durante a criação de um jogo, então é interessante buscar maneiras de automatizar o processo. Foi implementado um gerador de inimigos para um *vertical shooter* usando algoritmos genéticos, para avaliar a performance desse tipo algoritmo na solução do problema. Inimigos gerados se adaptam a diferentes perfis de jogadores, mas o elevado tempo de convergência dificulta a aplicação da tecnologia em jogos comerciais.

1 Introdução

A industria de jogos digitais passou e continua passando por uma rápida evolução quanto aos aspectos técnicos. Acompanhando essa evolução, cresceram drasticamente os custos de desenvolvimento de um jogo. Isso ocorreu principalmente pela demanda massiva de mão-de-obra especializada, necessária para o desenvolvimento de um projeto de larga escala. Um artigo[1] do site *kotaku* faz uma análise dos custos com o passar dos anos, com números ultrapassando as centenas de milhões de dólares (por projeto) na última década.

Além dos consideráveis custos monetários envolvidos no processo de desenvolvimento, existe também o problema do tempo. Jogos publicados pelos grandes *publishers*, os chamados jogos *triple A*, podem levar vários anos para serem desenvolvidos. Por exemplo, o jogo *Final Fantasy XV*, lançado em 2016, foi inicialmente anunciado em 2006, passando por um total de 10 anos de desenvolvimento.

Um fator de grande importância no desenvolvimento de jogos digitais é o balanceamento de dificuldade. Um jogo muito difícil pode ser frustrante para o jogador, enquanto um jogo fácil demais pode ter dificuldades em manter sua atenção.

Uma das formas de regular a dificuldade, é através do design de oponentes. Em muitos jogos, a dificuldade se dá através do confronto entre o jogador e oponentes com diferentes comportamentos. È trabalho do *game designer*, então, desenvolver oponentes que mantenham o jogador entretido durante sua experiência.

É importante notar, que as definições de dificuldade variam de jogador para jogador. Alguns jogadores podem ter experiência prévia de jogos semelhantes, enquanto outros são iniciantes. Para cada tipo de jogador, existe uma faixa de dificuldade(e consequentemente, um grupo de oponentes) ideal que maximiza o interesse e a diversão do jogador em um dado momento[2]. Essa faixa, denominada *flow zone* muda constantemente, conforme o jogador se acostuma com os controles e se adapta aos diferentes desafios.

É interessante então, buscar formas de automatizar o processo de criação e balanceamento de oponentes, de forma que cada jogador tenha uma experiência mais personalizada. O objetivo do projeto é então usar algoritmos genéticos para geração de oponentes adaptativos.

2 Referencial Teórico

Um Algoritmo genético é um algoritmo de otimização baseados na teoria da evolução e na seleção natural[3]. Algoritmos genéticos buscam determinar uma solução

ótima através da seleção e recombinação das melhores soluções entre várias iterações.

A execução de um algoritmo genético incia com a definição de uma população inicial, onde cada indivíduo possui uma série de genes que determinam suas características. Em cada geração (iteração do algoritmo), indivíduos mais aptos são selecionados de acordo com sua performance na solução do dado problema. Para determinar quais são as melhores soluções, é estabelecida uma *fitness function*, que quantifica a performance de cada membro da população.

Sendo determinadas as melhores soluções, são selecionados pares de indivíduos para a reprodução, dando preferência àqueles que tiveram melhor performance. Ocorre então uma recombinação de genes entre os pares selecionados, gerando nesse processo uma nova população. A transformação dos genes entre gerações pode ocorrer de duas maneiras:

- **Crossover**: Seleciona quais genes de cada um dos indivíduos pais serão usados para compor o indivíduos filho.
- **Mutação**: Genes de um indivíduo tem uma chance de serem alterados aleatoriamente.

Algoritmos evolutivos já foram aplicados na solução de vários problemas em jogos digitais, como na geração de *puzzles*[4] e na detecção de combos infinitos em jogos de luta[5]. A capacidade dessa classe de algoritmos de explorar diversas soluções, mesmo após a convergência, as tornam ferramentas de aplicação interessante na área já que os pontos ótimos, quando dependentes do jogador, estão em constante mudança.

No caso desse estudo, o foco é a geração dos inimigos. O uso de algoritmo genético para evolução de criaturas virtuais também já foi estudado, mesmo fora do contexto de jogos digitais[6].

3 Metodologia

3.1 Implementação do Jogo

Como base para implementação do algoritmo genético na geração e otimização de inimigos, foi implementado um jogo simples, no estilo *vertical shooter*. O jogo foi desenvolvido na *godot engine*[7], uma ferramenta *open-source* de desenvolvimento de jogos.

O jogo possui dois tipos de agentes: os inimigos e o jogador. Inimigos são criados na parte superior da tela, e movem-se no sentido do jogador, que se encontra na parte inferior. O jogador deve então se posicionar de forma a desviar de projéteis atirados pelos inimigos, e acerta-los com seus próprios. Inimigos que atingem a parte inferior da tela são considerados destruídos, para fins de progresso no jogo. Esse processo ocorre em várias iterações.

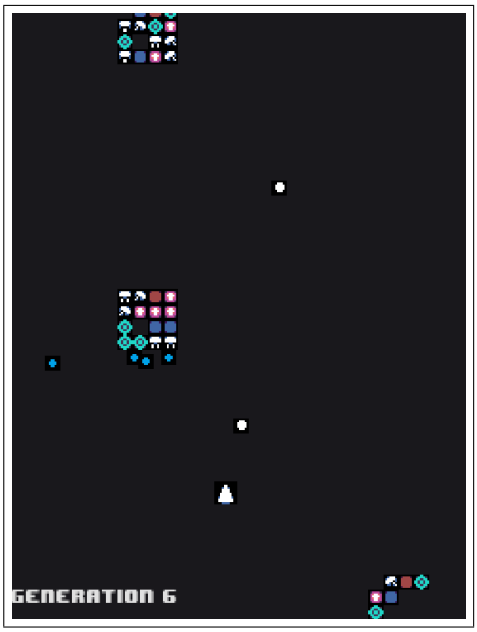


Figura 1 – *Screenshot* do jogo em funcionamento

Quando um certo número de inimigos são derrotados, é iniciada uma nova iteração. Cada iteração tem 3 momentos: inicia-se com a criação dos inimigos a partir de dados genes. Ocorre então o combate de inimigos contra o jogador, como descrito. Finalmente é usado o algoritmo genético para determinar os genes dos inimigos da próxima geração.

3.2 Modelagem do inimigo

Para facilitar a codificação em genes, inimigos foram criados de forma modular. Cada indivíduo é uma matriz de componentes, e todo funcionamento é delegado a esses componentes. Além disso, todo inimigo possui, obrigatoriamente, um núcleo. É apenas quando esse núcleo é destruído, que um inimigo é realmente derrotado.

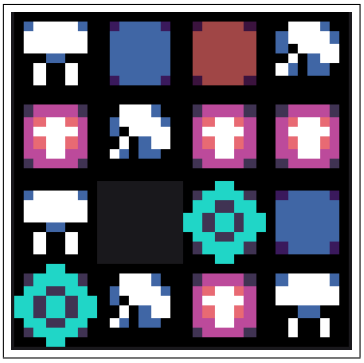


Figura 2 – Exemplo de um inimigo, como ele é representado no jogo. Cada bloco simboliza um componente

Componentes conferem diversas habilidades ao inimigo em questão, como por exemplo: capacidade de atirar projéteis, proteção contra projéteis do jogador, maior distância percorrida por projéteis. Cada componente possui pontos de vida independentes.

Quando esses são atingidos por tiros do jogador, seus pontos de vida são decrementados. Quando os pontos de vida chegam a zero, um componente é destruído, e não tem mais utilidade para o indivíduo. Quando o núcleo em particular é destruído, todos os outros componentes também são.

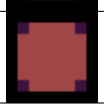
Componente	Funcionalidade
	Atiram projéteis em variados padrões
	Aumentam a velocidade de recarga das armas em 25%
	Aumenta a distância percorrida por projéteis em 30%
	Possui mais pontos de vida que os outros componentes, protegendo-os de projéteis do jogador
	Núcleo do inimigo

Tabela 1 – Componentes implementados

3.3 Algoritmo genético

Para aplicar o algoritmo genético, é necessário codificar os inimigos em sequências de genes. O genótipo de um indivíduo é composto de 2 partes: uma que guarda a posição do núcleo, e outra que guarda a disposição dos demais componentes. Essa separação é necessária para impedir que um inimigo fique com múltiplos núcleos (ou nenhum) após o *crossover*.

A posição do núcleo é armazenada em um simples vetor de 2 elementos, sendo esses a posição x e y do núcleo na matriz do inimigo. A parte do genótipo que armazena os outros componentes é uma matriz. Cada posição dessa matriz guarda um número inteiro, que serve de identificador para um tipo de componente. São considerados genes individuais cada elemento da matriz de componentes, além da vetor de posição do núcleo.

O *crossover* entre dois genótipos ocorre de forma uniforme: para cada gene, existe 50% de chance de escolha para ambos os pais. Além disso, existe uma chance de mutação independente para cada gene.

Para determinar o indivíduos mais aptos em cada geração, a *fitness function* usada foi a função(1):

$$F(T, C_{total}, C_{alive}, H, P, d) = (T * 5) + (\frac{C_{total}}{C_{alive}} * 50 * (1 - \frac{1}{max(H^3, 1)})) + P + (\sum_{i=0}^{10} \frac{5}{d_i^2}) \quad (1)$$

Onde T representa o tempo de vida total de um inimigo, C_{total} é a quantidade de componentes total de um inimigo, C_{alive} é a quantidade de componentes vivos quando um inimigo consegue atingir a parte inferior da tela, H é o número de vezes que um inimigo foi atingido, P é o número de vezes que o inimigo atingiu o *player* com seus projéteis e d é um vetor ordenado com as distâncias mínimas para o jogador de cada projétil atirado. Os pesos e os parâmetros da função foram determinados de forma empírica.

Sendo atribuídas as pontuações, a seleção de pares ocorre através do método da roleta. O método da roleta consiste em uma seleção aleatória de indivíduos, onde aqueles com maior *fitness* tem maior chance de serem escolhidos[8].

4 Resultados

Para realizar testes do jogo e do algoritmo, foram implementadas 2 IAs(inteligências artificiais). A primeira, que será referida como *AimingBot*, busca eliminar inimigos da maneira mais rápida o possível, mirando diretamente em seus núcleos. Já a segunda, referida como *DirectionalBot* , não leva em consideração o estado dos inimigos, se movendo fixamente em uma direção até encontrar o limite da tela. Quando isso ocorre, ela muda a direção do jogador. A ideia é simular diferentes tipos de jogadores, e assim observar se os inimigos se adaptam para os dois.

4.1 Testes

Para testar a velocidade de convergência na evolução, foi medida a fitness média de cada geração por 150 gerações. Foi fixado então o tamanho da geração em 15 indivíduos, e variada a taxa de mutação. Esse teste foi feito com a IA *AimingBot*. Os resultados podem ser observados na Fig. 3.

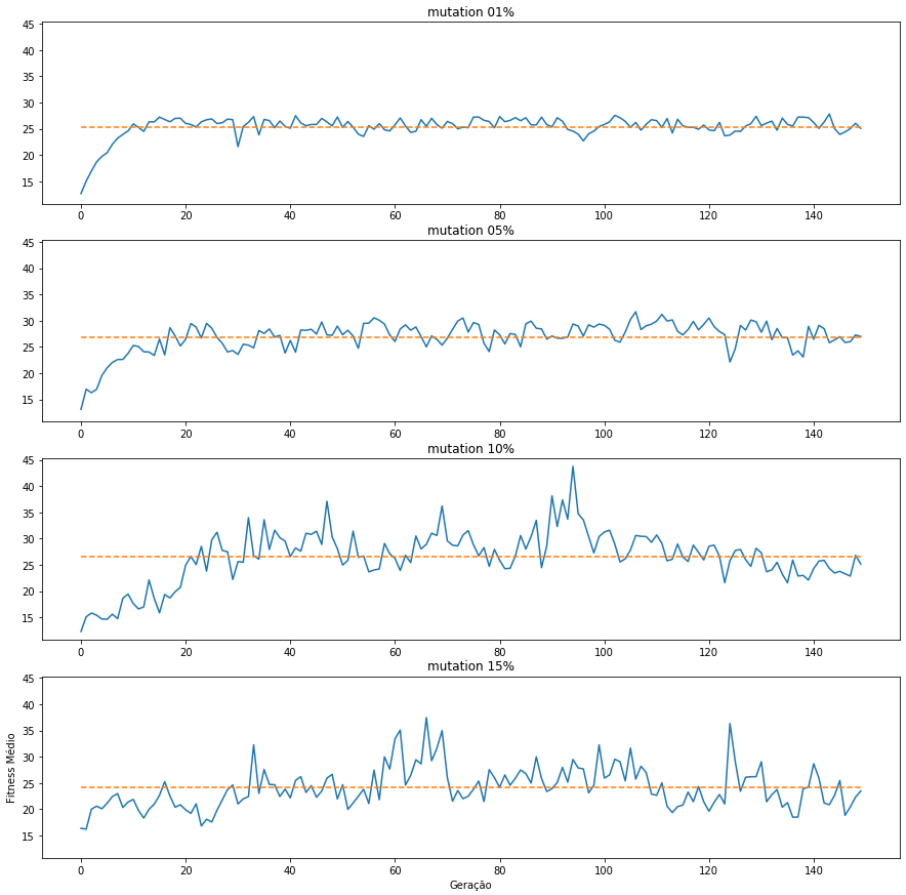


Figura 3 – Teste de convergência com variação na taxa de mutação

Percebe-se que taxas de mutação menores como 1% e 5% convergem mais rápido e são mais estáveis, mantendo-se na média por grande parte das gerações. Já as taxas maiores, conseguem atingir resultados melhores em certos pontos, só que esses resultados são altamente instáveis. A fitness média não varia muito entre as instancias, porém ela é maior nos testes com 5% e 10% de taxas de mutação.

Na Fig.4, foi fixada a taxa de mutação em 2% e variado o tamanho da população. Para populações maiores, o tempo de convergência é consideravelmente maior. Porém, a fitness média atingida após a convergência também cresce com o tamanho da população.

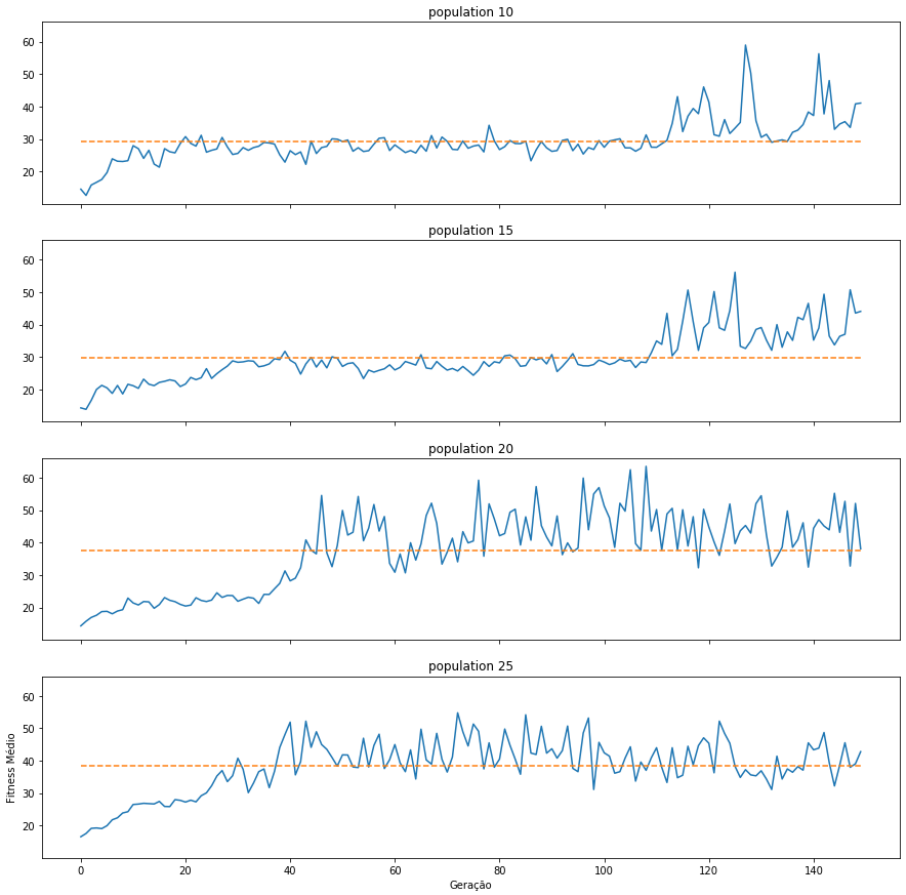


Figura 4 – Teste de convergência com variação no tamanho da população

Para simular o comportamento do algoritmo com diferentes perfis de jogadores, foi feita uma comparação entre o *AimingBot* e o *DirectionalBot*, como observado na Fig. 5. Nesse teste, foi usada uma taxa de mutação de 2% e uma população de tamanho 15. O teste com o *AimingBot* leva mais gerações para convergir, e a fitness média é mais baixa. Isso indica que, como esperado, o *AimingBot* é um melhor oponente para os inimigos gerados.

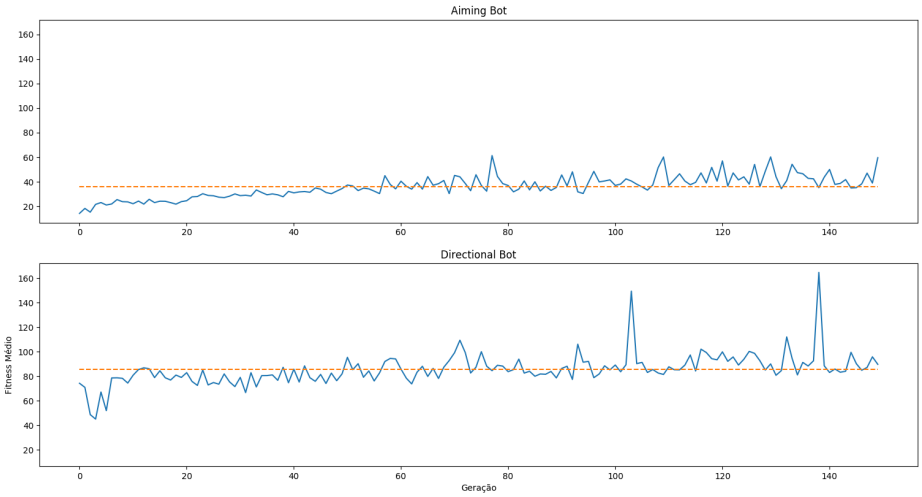


Figura 5 – Teste de convergência para as duas IAs

5 Conclusão

É visível que os inimigos gerados se adaptam, desenvolvendo diferentes estratégias para enfrentar jogadores com comportamento distintos. Nas Figs. 6 e 7 é possível observar características que evoluiram naturalmente para retaliar o comportamento das IAs. Como o *AimingBot* busca sempre destruir o núcleo do inimigo, foi evoluída uma camada protetora de módulos defensivos na frente do núcleo. No caso do *DirectionalBot*, é improvável que o núcleo seja atingido, então o inimigo busca proteger seus outros componentes, preenchendo a linha mais baixa da matriz com escudos.

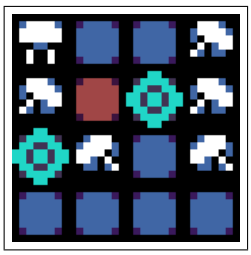


Figura 6 – Inimigo evoluído nos testes com o *DirectionalBot*

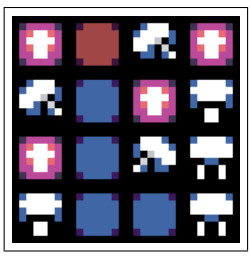


Figura 7 – Inimigo evoluido nos testes com o *AimingBot*

Concluí-se então que existe potencial na aplicação de algoritmos genéticos na

geração procedural e no balanceamento automático de dificuldade. Porém, pode ser difícil aplicar esse método em jogos digitais comerciais, devido ao longo tempo necessário para a convergência. É impensável fazer o jogador enfrentar diversas gerações de oponentes disfuncionais até que sejam evoluídos inimigos desafiadores. Esse fator também limita o tamanho da população, já que maiores populações implicam em um maior tempo de jogo por geração.

Em alguns casos, é possível que ocorram degenerações no resultado. Isso ocorre devido a natureza da *fitness function*, que poder retornar valores diferentes para o mesmo indivíduo, dependendo do estado do jogador.

Para trabalhos futuros, existem vários estudos interessantes possíveis:

- Estudar como mudanças de comportamento durante o jogo afetam a pontuação de *fitness*, e observar se os inimigos se adaptam de forma dinâmica.
- Nesse estudo,os inimigos se comportam de maneira extremamente simples e previsível. Seria interessante combinar a geração dos corpos com alguma forma de inteligência artificial adaptativa.

Referências

[1] How Much Does It Cost To Make A Big Video Game? <https://kotaku.com/how-much-does-it-cost-to-make-a-big-video-game-1501413649>

[2] GameFlow: A Model for Evaluating Player Enjoyment in Games <https://kotaku.com/how-much-does-it-cost-to-make-a-big-video-game-1501413649>

[3] A genetic algorithm tutorial <https://doi.org/10.1007/BF00175354>

[4] Evolving lock-and-key puzzles based on nonlinear player progression and level exploration <https://www.sbgames.org/sbgames2018/files/papers/ComputacaoShort/188344.pdf>

[5] Discovering Combos in Fighting Games with Evolutionary Algorithms <https://homepages.dcc.ufmg.br/~chaimo/public/Gecco16>

[6] Evolving virtual creatures <https://https://dl.acm.org/citation.cfm?id=192167>

[7] Godot Engine Homepage. <https://godotengine.org/>

[8] Review of Selection Methods in Genetic Algorithms <https://www.ijecs.in/index.php/ijecs/article/download/2562/2368/>